US006545981B1

(12) **United States Patent**
Garcia et al.

(10) **Patent No.: US 6,545,981 B1**
(45) **Date of Patent: Apr. 8, 2003**

(54) **SYSTEM AND METHOD FOR IMPLEMENTING ERROR DETECTION AND RECOVERY IN A SYSTEM AREA NETWORK**

(75) Inventors: **David J. Garcia**, Los Gatos; **Richard O. Larson**, Cupertino, both of CA (US); **Stephen G. Low**; **William J. Watson**, both of Austin, TX (US)

(73) Assignee: **Compaq Computer Corporation**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/224,115**

(22) Filed: **Dec. 30, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/070,650, filed on Jan. 7, 1998.

(51) **Int. Cl.**[7] **H04L 12/56**

(52) **U.S. Cl.** **370/242**; 370/248; 370/392; 714/48

(58) **Field of Search** 370/216, 217, 370/225, 252, 228–230, 242, 248, 257, 389, 394, 400, 465, 392; 340/2.1, 2.7, 3.43; 709/107, 239, 250, 238; 714/100, 48, 21; 379/221.01, 221.04

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,733,350 A		3/1988	Tone et al.	364/200
5,193,149 A	*	3/1993	Awiszio et al.	395/200
5,386,524 A		1/1995	Lary et al.	395/400
5,555,405 A		9/1996	Griesmer et al.	395/600
5,563,879 A		10/1996	Sanders et al.	370/60
5,574,849 A		11/1996	Sonnier et al.	395/182.1
5,619,274 A		4/1997	Roop et al.	348/461
5,675,579 A	*	10/1997	Watson et al.	370/248
5,678,007 A		10/1997	Hurvig	395/200.12
5,706,281 A	*	1/1998	Hashimoto et al.	370/352
5,737,595 A		4/1998	Cohen et al.	395/61
5,802,050 A		9/1998	Petersen et al.	370/394
5,920,886 A		7/1999	Feldmeier	711/108
5,991,797 A		11/1999	Furtral et al.	709/216
6,119,244 A		9/2000	Schoenthal et al.	714/4
6,272,591 B2		8/2001	Grun	711/114
6,347,337 B1		2/2002	Shah et al.	709/224
2002/0029305 A1		3/2002	Satran et al.	710/33

FOREIGN PATENT DOCUMENTS

EP	0 757 318 A2	2/1997	G06F/13/12

OTHER PUBLICATIONS

Garcia et al., "ServerNet II" *Parallel Computer Routing and Communication*, (2nd Int. WKSP), Jun. 26, 1997, pp. 119–135, XP002103164, Atlanta, GA.

Eicken Von T. et al., "U–Net: A User–Level Network Interrface for Parallel and Distributed Computing," *Operating Systems Review* (SIGOPS), vol. 29, No. 5, Dec. 1, 1995, pp. 40–53.

Dunning D. et al., "The Virtual Interface Architecture," *IEEE Micro*, vol. 18, No. 2, 3/98, pp. 66–76.

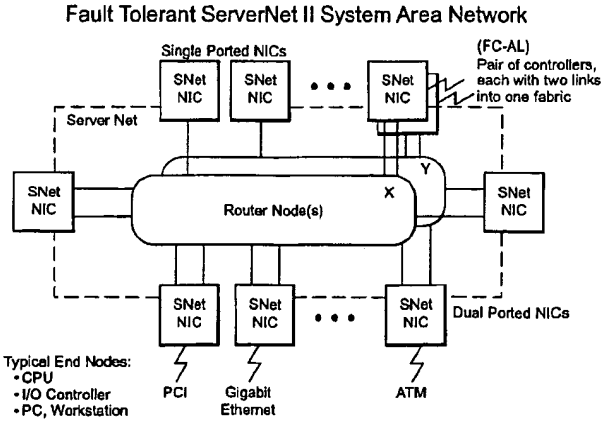
* cited by examiner

Primary Examiner—Huy D. Vu
Assistant Examiner—Duc Ho
(74) *Attorney, Agent, or Firm*—Oppenheimer, Wolff & Donnelly, LLP; Leah Sherry

(57) **ABSTRACT**

A system and method for facilitating both in-order and out-of-order packet reception in a SAN includes requestor and responder nodes, coupled by a plurality of paths, that maintain the good and bad status of each path and also maintain local copies of a message sequence number. If an error occurs for a transaction over a given path, the requestor informs the responder, over a good path, that the given path has failed and both nodes update their path status to indicate that the given path is bad. A barrier transaction is used by the requestor to determine whether the error is transient or permanent, and, if the error is transient, the requestor retries the transaction.

22 Claims, 6 Drawing Sheets



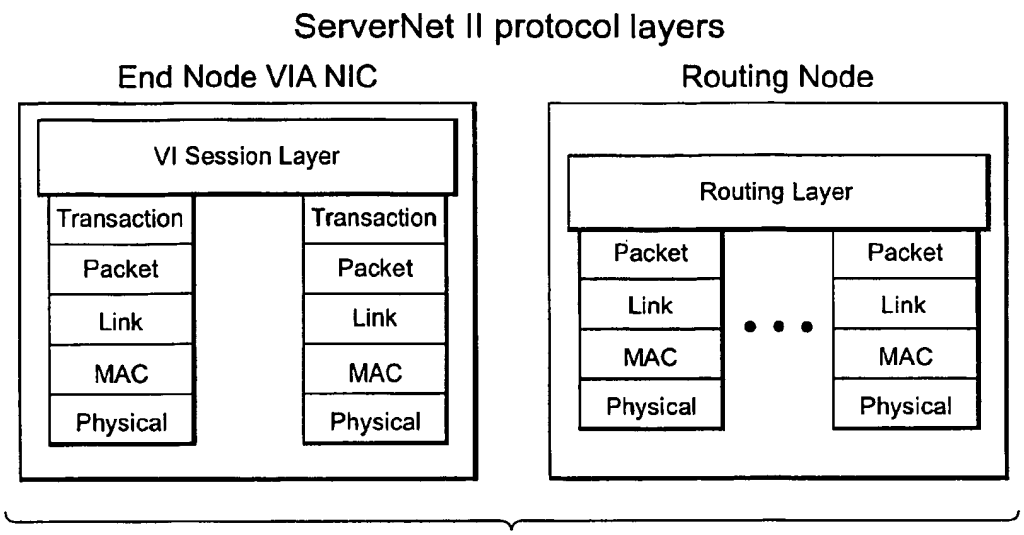

ServerNet II protocol layers
End Node VIA NIC
Routing Node
VI Session Layer
Transaction
Packet
Link
MAC
Physical
Transaction
Packet
Link
MAC
Physical
Routing Layer
Packet
Link
MAC
Physical
Packet
Link
MAC
Physical

FIG. 1

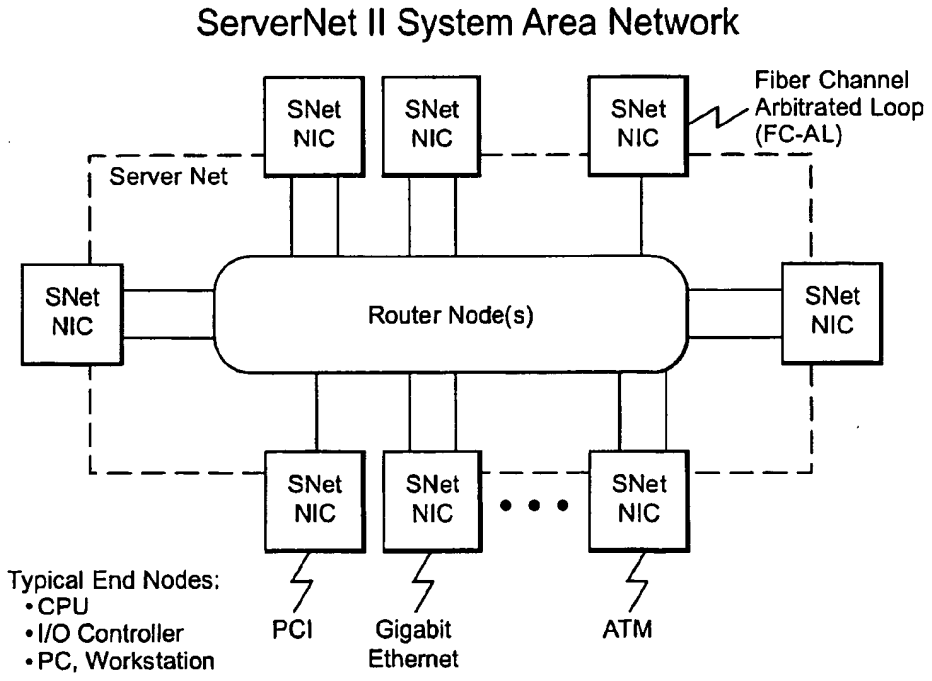

ServerNet II System Area Network
SNet NIC
SNet NIC
SNet NIC
Fiber Channel Arbitrated Loop (FC-AL)
Server Net
SNet NIC
Router Node(s)
SNet NIC
SNet NIC
SNet NIC
SNet NIC
Typical End Nodes:
• CPU
• I/O Controller
• PC, Workstation
PCI
Gigabit Ethernet
ATM

FIG. 2

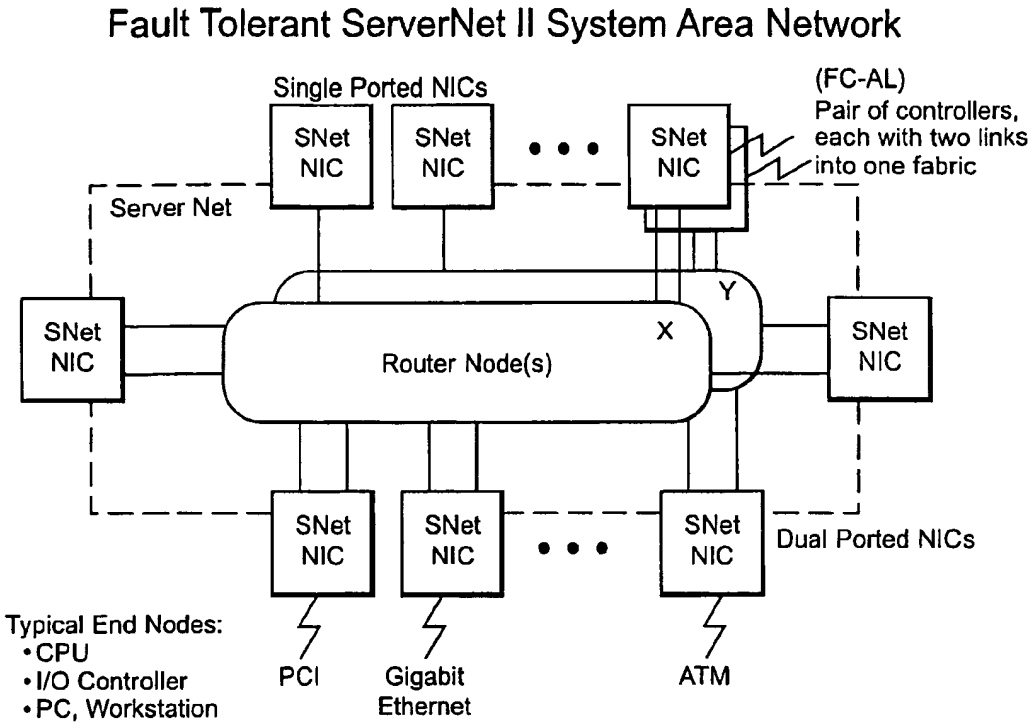

Fault Tolerant ServerNet II System Area Network
Single Ported NICs
SNet NIC
SNet NIC
SNet NIC
(FC-AL)
Pair of controllers, each with two links into one fabric
Server Net
Y
X
SNet NIC
Router Node(s)
SNet NIC
SNet NIC
SNet NIC
SNet NIC
Dual Ported NICs
Typical End Nodes:
•CPU
•I/O Controller
•PC, Workstation
PCI
Gigabit Ethernet
ATM

FIG. 3

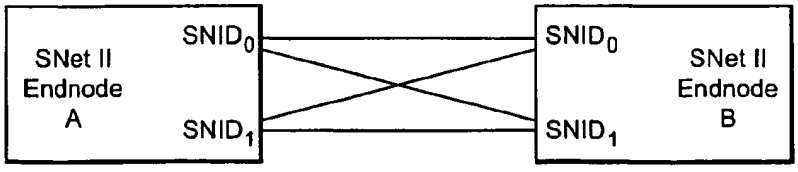

SNet II Endnode A
SNID0
SNID1
SNID0
SNID1
SNet II Endnode B
Four Logical Paths between SNet II Endnodes

FIG. 4

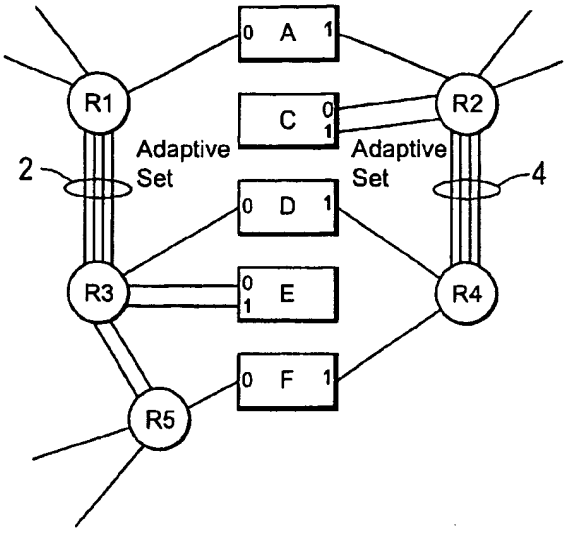


FIG. 5

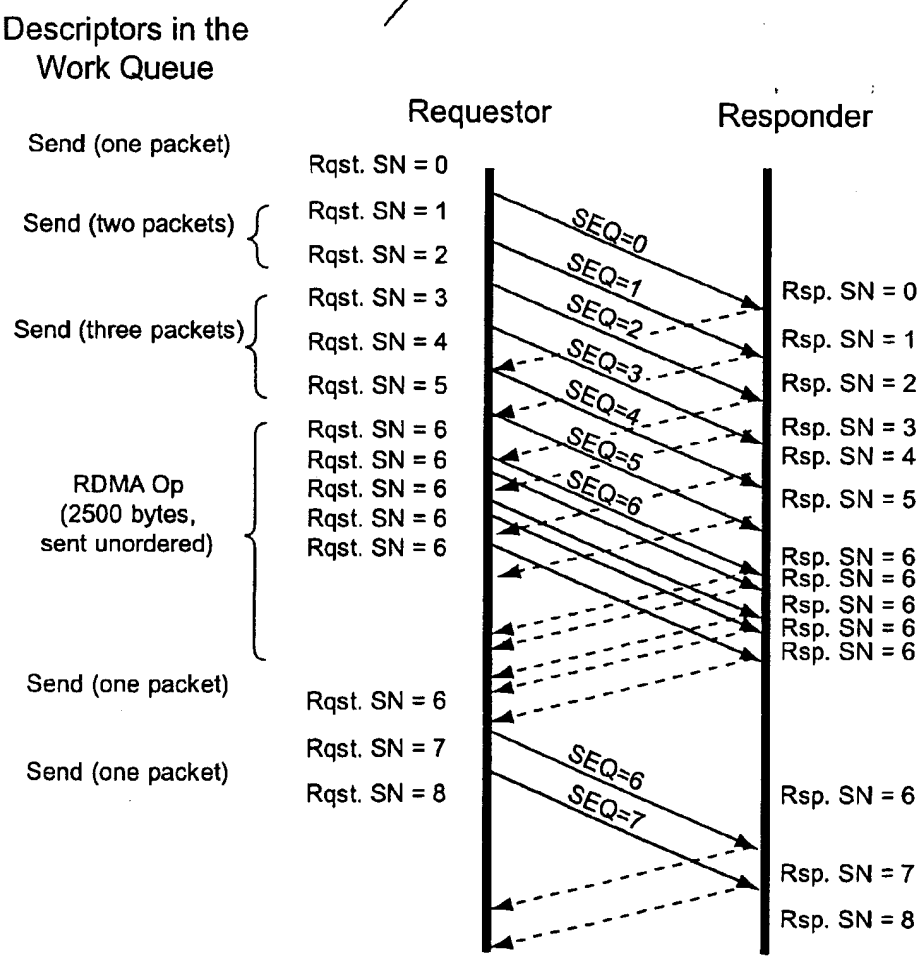


FIG. 6 Sequence Number Example

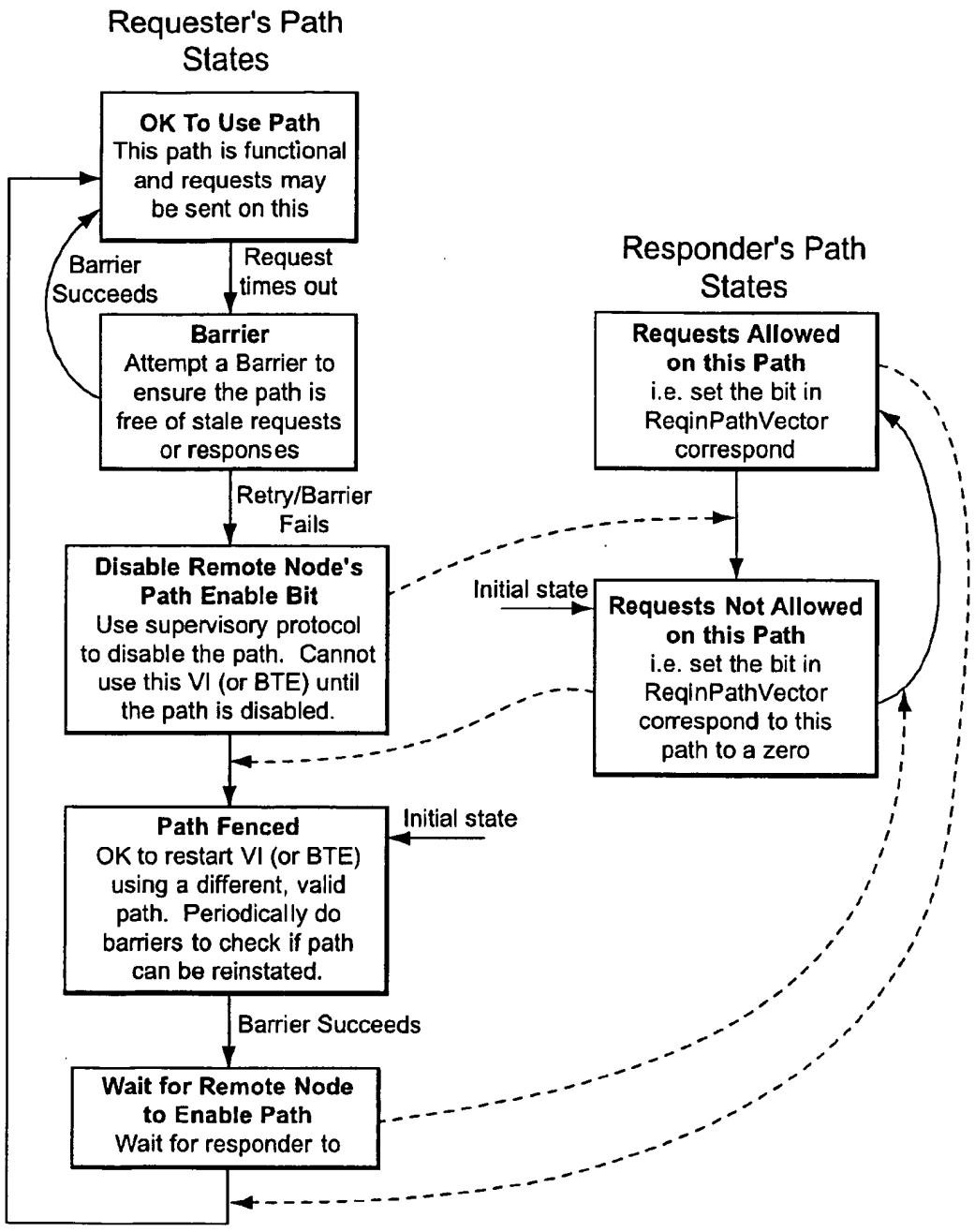
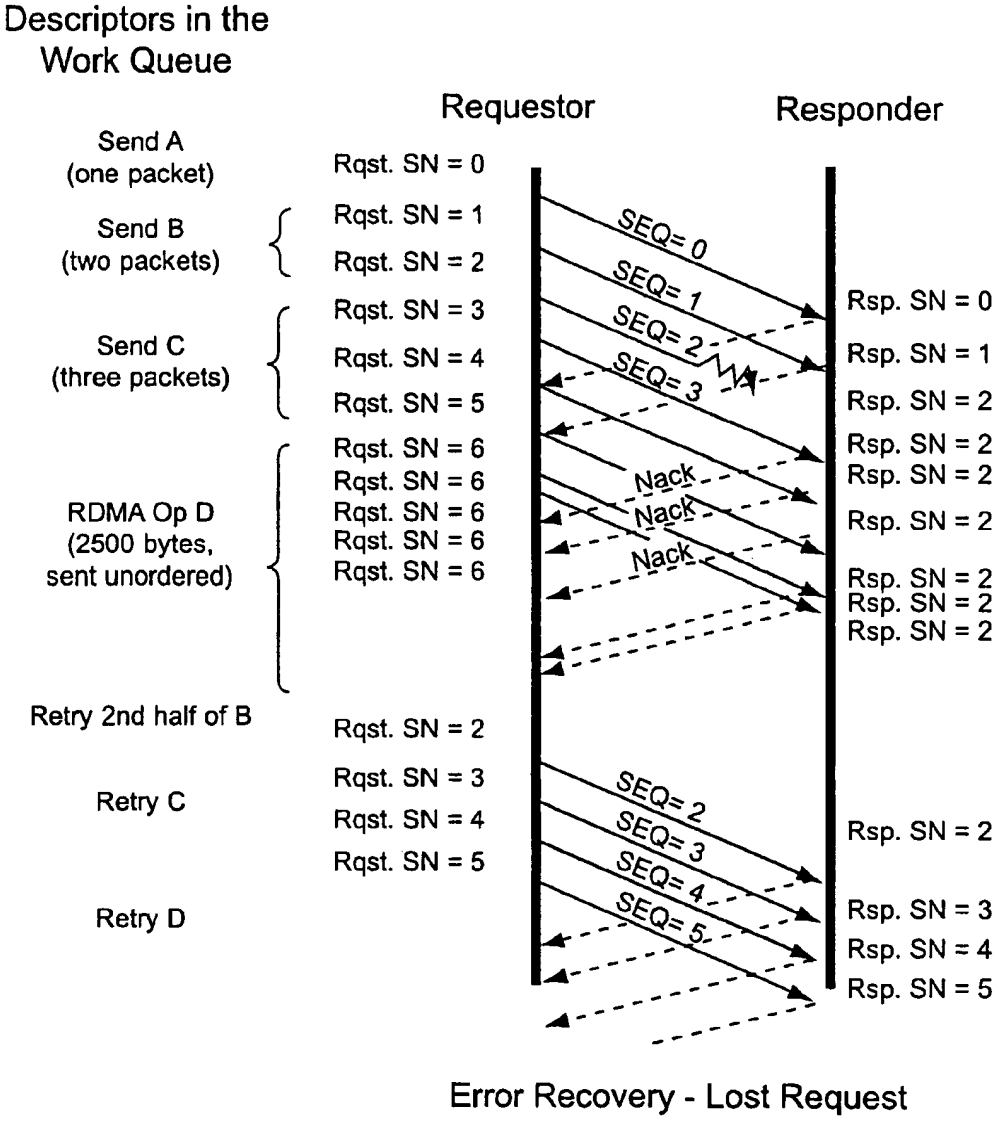
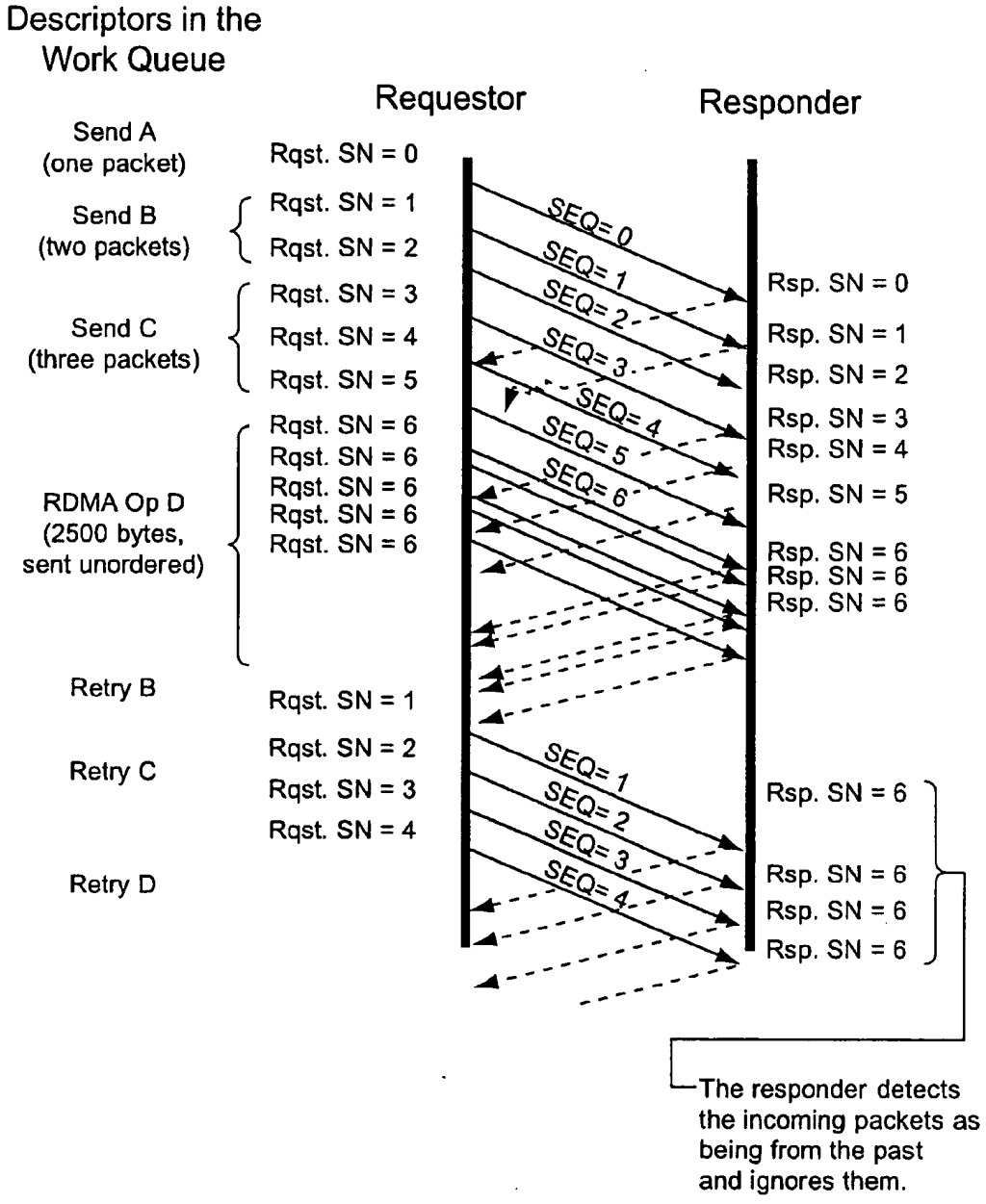


FIG. 7
Path State Diagram



Error Recovery - Lost Request

FIG. 8



Error Recovery - Lost Acknowledgment

FIG. 9

SYSTEM AND METHOD FOR IMPLEMENTING ERROR DETECTION AND RECOVERY IN A SYSTEM AREA NETWORK

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims priority from Provisional App. No. 60/070,650, filed Jan. 7, 1998, which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

Traditional network systems utilize either channel semantics (send/receive) or memory semantics (DMA) model. Channel semantics tend to be used in I/O environments and memory semantics tend to be used in processor environments.

In a channel semantics model, the sender does not know where data is to be stored, it just puts the data on the channel. On the sending side, the sending process specifies the memory regions that contain the data to be sent. On the receiving side, the receiving process specifies the memory regions where the data will be stored.

In the memory semantics model, the sender directs the data to a particular location in the memory, utilizing remote direct memory access (RDMA) transactions. The initiator of the data transfer specifies both the source buffer and destination buffer of the data transfer. There are two types of RDMA operations, read and write.

The virtual interface architecture (VIA) has been jointly developed by a number of computer and software companies. VIA provides consumer processes with a protected, directly accessible interface to network hardware, termed a virtual interface. VIA is especially designed to provide low latency message communication over a system area network (SAN) to facilitate multi-processing utilizing clusters of processors.

A SAN is used to interconnect nodes within a distributed computer system, such as a cluster. The SAN is a type of network that provides high bandwidth, low latency communication with a very low error rate. SANs often utilize fault-tolerant capability.

It is important for the SAN to provide high reliability and high-bandwidth, low latency communication to fulfill the goals of the VIA. Further, it is important for the SAN to be able to recover from errors and continue to operate in the event of equipment failures. Error recovery must be accomplished without high CPU overhead associated with all transactions. Furthermore, error recovery should not increase the complexity for the consumer of VIA services.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, a SAN maintains local copies of a sequence number for each data transfer transaction at the requester and responder nodes. Each data transfer is implemented by the SAN as a sequence of request/response packet pairs. An error condition arises if a response to any request packet is not received at the requesting node. The responder and requestor nodes are coupled by a plurality of paths and each node maintains a record of the good or bad status of each path. If a transaction fails and the path is permanently bad, both nodes update their status to indicate that the path is bad. This is to prevent further transactions from including any stale requests that are potentially still in the network from arriving at the destination and potentially corrupting data.

According to another aspect of the invention, if an error occurs on a path the requestor node implements a barrier transaction on the path to determine if the failure is permanent or transient.

According to another aspect of the invention, the barrier transaction is performed by writing a number chosen from a large number space in a way that minimizes the probability of reusing the number in a short period of time.

According to one aspect of the invention, the number is randomly chosen from a large number space.

According to another aspect of the invention, the large number is based on the requestor ID and an incrementing component managed by the requestor.

According to another aspect of the invention, if the failure is transient the requestor retransmits packets starting with the packet that first caused an error condition to be detected.

According to another aspect of the invention, a sequence number is included in each request packet and copied into each response packet. A local copy of the sequence number is maintained at the requestor and responder nodes. If the sequence number in the request packet does not match the sequence number at the responder, a negative acknowledge response packet is generated.

Other features and advantages of the invention will be apparent in view of the following detailed description and appended drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram depicting ServerNet protocol layers implemented by hardware, where ServerNet is a SAN manufactured by the assignee of the present invention;

FIGS. **2** and **3** are block diagrams depicting SAN topologies;

FIG. **4** is a schematic diagram depicting logical paths between end nodes of a SAN;

FIG. **5** is a schematic diagram depicting routers and links connecting SAN end nodes;

FIG. **6** is a graph depicting the transmission of request and response packets between a requestor and a responder end node. FIG. **6** shows the sequence numbers used in packets for three Send operations, an RDMA operation, and two additional Send operations. The diagram shows the sequence numbers maintained in the requestor logic, the sequence number contained in each packet, and the sequence numbers maintained at the responder logic;

FIG. **7** is two interlocked state diagrams showing the state that software on the requestor and responder moves through for each path;

FIG. **8** is a graph depicting retransmission during error recovery due to a lost request packet; and

FIG. **9** is a graph depicting retransmission during error recovery due to a lost acknowledgment packet.

DESCRIPTION OF THE EMBODIMENTS

The preferred embodiments will be described implemented in the ServerNet II (ServerNet) architecture, manufactured by the assignee of the present invention, which is a layered transport protocol for a System Area Network (SAN) optimized to support the Virtual Interface (VI) architecture session layer which has stringent user-space to user-space latency and bandwidth requirements. These requirements mandate a reliable hardware (HW) message transport solution with minimal software (SW) protocol stack overhead. The ServerNet II protocol layers for an end

node VI Network Interface controller/Card (NIC) and for a routing node are illustrated in FIG. 1. A single NIC and VI session layer may support one or two ports, each with its associated transaction, packet, link-level, MAC (media access) and physical layer. Similarly, routing nodes with a common routing layer may support multiple ports, each with its associated link-level, MAC and physical layer.

Support for two ports enables the ServerNet II SAN to be configured in both non-redundant and redundant (fault tolerant, or FT) SAN configurations as illustrated in FIG. 2 and FIG. 3. On a fault tolerant network, a port of each end node may be connected to each network to provide continued VI message communication in the event of failure of one of the SANs. In the fault tolerant SAN, nodes may be ported into a single fabric or single ported end nodes may be grouped into pairs to provide duplex FT controllers. The fabric is the collection of routers, switches, connectors, and cables that connect the nodes in a network.

The following describes general ServerNet II terminology and concepts. The use of the term "layer" in the following description is intended to describe functionality and does not imply gate level partitioning.

Two ports are supported on a NIC for both performance and fault tolerance reasons. Both of these ports operate under the same session layer VI engine. That is, data may arrive on any port and be destined for any VI. Similarly, the VIs on the end node can generate data for any of these ports.

ServerNet II packets are comprised of a series of data symbols followed by a packet framing command. Other commands, used for flow control, virtual channel support, and other link management functions, may be embedded within a packet. Each request or response packet defines a variety of information for routing, transaction type, verification, length and VI specific information.

i. Routing in the ServerNet II SAN is destination-based using the first 3 bytes of the packet. Each NIC end node port in the network is uniquely defined by a 20 bit Port SNID (ServerNet Node ID). The first 3 bytes of a packet contain the Destination port's SNID or DID (destination port ID) field, a three bit Adaptive Control Bits (ACB) field and the fabric ID bit. The ACB is used to specify the path (deterministic or link-set adaptive) used to route the packet to its destination port as described in the following section.

ii. The transaction type fields define the type of session layer operation that this ServerNet II packet is carrying and other information such as whether it is a request or a response and, if a response, whether it is an ACK (acknowledgment) or a NACK (negative acknowledgment). The ServerNet II SAN also supports other transaction types.

iii. Transaction verification fields include the source port ID (SID) and a Transaction Serial Number. The transaction serial number enables a port with multiple requests outstanding to uniquely match responses to requests.

iv. The Length field consists of an encoding of the number of bytes of payload data in the packet. Payloads up to 512 bytes are supported and code space is reserved for future increases in payload size.

v. The VI Session Layer specific fields describe VI information such as the VI Operation, the VIA Sequence number and the Virtual Interface.ID number. The VI Operation field defines the type of VI transaction being sent (Send, RDMA Read, RDMA Write) and other control information such as whether the packet is ordered or unordered, whether there is immediate data and/or whether this is the first or last packet in a session layer multi-packet transfer. Based on the VI transaction type and control information, a 32 bit Immediate data field or a 64 bit Virtual address may follow the VI ID number.

vi. The payload data field carries up to 512 bytes of data between requesters and responders and may contain a pad byte.

vii. The CRC field contains a checksum computed over the entire packet.

Transaction Overview

The basic flow of transactions through the ServerNet II SAN will now be described. VI requires the support of Send, RDMA read and RDMA write transactions. These are translated by the VI session layer into a set of ServerNet II transactions (request/response packet pairs). All data transfers (e.g., reading a disk file to CPU memory, dumping large volumes of data from a disk farm directly over a high-speed communications link, one end node simply interrupting another) consist of one or more such transactions.

Creating a Request Packet

The VI User Agent provides the low level routines for VIA Send, RDMA Write, and RDMA Read operations. These routines place a descriptor for the desired transfer in the appropriate VI queue and notify the VIA hardware that the descriptor is ready for processing. The VIA hardware reads the descriptor, and based on the descriptor contents, builds the ServerNet request packet header and assembles the data payload (if appropriate).

Dual Ports and Ordering

In a NIC with two ports, it is possible for a single VIA interface to process Sends and RDMA operations from several different VIs in parallel. It is also possible for a large RDMA transfer from a single VI to be transferred on both of the ports simultaneously. This latter feature is called Multi-pathing.

ServerNet II end nodes can connect both their ports to a single network fabric so that there are up to four possible paths between ServerNet II end nodes. Each port of a single end node may have a unique ServerNet ID (SNID). FIG. 4 depicts the four possible paths that End node A can use when sending request to End node B:

1) End node A SNID[0] to End node B SNID[0]
2) End node A SNID[0] to End node B SNID[1]
3) End node A SNID[l] to End node B SNID[0]
4) End node A SNID[1] to End node B SNID[1]

FIG. 5 depicts a network topology utilizing routers and links. In FIG. 5, end nodes A–F, each having first and second send receive ports 0 and 1, are coupled by a ServerNet topology including routers R1–R5. Links are represented by lines coupling ports to routers or routers to routers. A first adaptive set (fat pipe) 2 couples routers RI and R3 and a second adaptive set (fat pipe) 4 couples routers R2 and R4.

Routing may be deterministic or link set adaptive. An adaptive link-set is a set of links (also called lanes) between two routers that have been grouped to provide higher bandwidth. The Adaptive Control Bits (ACB) specify which type of routing is in effect for a particular packet.

Deterministic routing preserves strict ordering for packets sent from a particular source port to a destination port. In deterministic routing, the ACB field selects a single path or

lane through an adaptive link-set. Send transactions for a particular VI require strict ordering and therefore use deterministic routing.

RDMA transactions, on the other hand, may make use of all possible paths in the network without regard for the ordering of packets within the transaction. These transactions may use link-set adaptive routing as described below. The ACB field specifies which specific link (or lane) in this link-set is to be used for deterministic routing.

Alternatively, the ACB field can specify link-set adaptivity which enables the packets to dynamically choose from any of the links in the link-set.

A sample topology with several different examples of multipathing using link and path adaptivity is shown in FIG. 5.

Multipathing allows large block transfers done with RDMA Read or Write operations to simultaneously use both ports as well as adaptive links between the two communicating NICs. Since the data transfer characteristics of any one VI are expected to be bursty, multipathing allows the end node to marshal all its resources for a single transfer. Note that multipathing does not increase the throughput of multiple Send operations from one VI. Sends from one VI must be sent strictly ordered. Since there are no ordering guarantees between packets originating from different ports on a NIC, only one port may be used per Send. Furthermore, only a single ordered path through the Network may be used, as described in the following.

Transaction and Packet Layers

The transaction layer builds the ServerNet II request packet by filling in the appropriate SID, Transaction Serial Number (TSN), and CRC. The SID assigned to a packet always corresponds to the SNID of the port the packet originates from. The TSN can be used to help the port manage multiple outstanding requests and match the resulting responses uniquely to the appropriate request. The CRC enables the data integrity of the packet to be checked at the end node and by routers enroute.

Following the ServerNet II link protocol, the packet is encoded in a series of data symbols followed by a status command. The ServerNet II link layer uses other commands for flow control and link management. These commands may be inserted anywhere in the link data stream, including between consecutive data symbols of a packet. Finally, the symbols are passed through the MAC layer for transmission on the physical media to an intermediate routing node.

Routing

The routing control function is programmable so that the packet routing can be changed as needed when the network configuration changes (e.g., route to new end nodes). Router nodes serve as crossbar switches; a packet on any incoming (receive) side of a link can be switched to the outgoing (transmit) side of any link. As the incoming request packet arrives at a router node, the first three bytes, containing the DID and ACB fields, are decoded and used to select a link leading to the destination node. If the transmit side of the selected link is not busy, the head of the packet is sent to the destination node whether or not the tail of the packet has arrived at the routing node. If the selected link is busy with another packet, the newly arrived packet must wait for the target port to become free before it can pass through the crossbar.

As the tail of the packet arrives, the router node checks the packet CRC and updates the packet status (good or bad). The packet status is carried by a link symbol TPG (this packet good) or TPB (this packet bad) appended at the end of the packet. Since packet status is checked on each link, a packet status transition (good to bad) can be attributed to a specific link. The packet routing process described above is repeated for each router node in the selected path to the destination node.

Receiving a Request Packet

When the request packet arrives at the destination node, the ServerNet II interface receiver checks its validity (e.g., must contain correct destination node ID, the length is correct, the Fabric bit in the packet matches the Fabric bit associated with the receiving port, the request field encodes a valid request, and CRC must be good,). If the packet is invalid for any reason, the packet is discarded. The ServerNet II interface may save error status for evaluation by software. If these validity checks succeed, several more checks are made. Specifically, if the request specifies an RDMA Read or Write, the address is checked to ensure access has been enabled for that particular VI. Also, the input port and Source ID of the packet are checked to ensure access to the particular VI is allowed on that input port from the particular Source. If the packet is valid, the request can be completed.

Response Packet

A response is created based on the success (ACK response) or failure (NACK response) of the request packet. A successful read request, for example, would include the read data in the ACK response. The source node ID from the request packet is used as the destination node ID for the response packet. The response packet must be returned to the original source port. The path taken by the response is not necessarily the reverse of the path taken by the request. The network may be configured so that responses take very different paths than requests. If strict ordering is not required, the response, like the request, may use link-set adaptivity. The response packet is routed back to the SNID specified by the SID field of the request. The ACB field of the request packet is also duplicated for the response packet.

The response can be matched with the request using the TSN and the packet validity checks. If an ACK response passes these tests, the transaction layer passes the response data to the session layer, frees resources associated with the request, and reports the transaction as complete. If a NACK response passes these tests, the end node reports the failure of the transaction to the session layer. If a valid ACK/NACK response is not received within the allotted time limit, a time-out error is reported.

The requestor can stream many strictly ordered ServerNet II messages onto the wire before receiving an acknowledgment. The sliding window protocol allows the requestor to have up to 128 packets outstanding per VI.

The hardware can operate in one of two modes with respect to generating multiple outstanding request packets:

1. The hardware can stream packets from the same VI send queue onto the wire, and start the next descriptor before receiving all the acknowledgments from the current descriptor. This is referred to as "Next Descriptor After Launch" or NDAL.

2. The hardware can stream packets to a single descriptor onto the wire but wait for all the outstanding acknowledgments to complete before starting the next descriptor. This is referred to as "Next Descriptor after ACK" or NDAA.

The choice of NDAL or NDAA modes of operation is determined by how strongly ordered the packets are generated.

Ordered and unordered messages may be mixed on a single VI. When generating an unordered message, the requester must wait for completion of all acknowledgments to unordered packets before starting the next descriptor.

Ordering of Send Packets Presented to Transaction Layer

The VI architecture has no explicit ordering rules as to how the packets that make up a single descriptor are ordered among themselves. That is, VIA only guarantees the message ordering the client will see. For example, VIA requires that Send descriptors for a particular VI be completed in order, but the VIA specification does not say how the packets will proceed on the wire.

The ServerNet II SAN requires that all Send packets destined for a particular VI be delivered by the SAN in strict order. As long as deterministic routing is used, the network assures strict ordering along a path from a particular source node to a particular destination node. This is necessary because the receiving node places the incoming packets into a scatter list. Each incoming packet goes to a destination determined by the sum total of bytes of the previous packets. The strict ordering of packets is necessary to preserve integrity of the entire block of data being transferred because incoming packets are placed in consecutive locations within the block of data. Each packet has a sequence number to allow the receiver to detect an out of order, missing, or repeated packet.

There are two ways for an end node to meet these ordering requirements:

a. The end node can wait for the acknowledgment from each Send packet to complete before starting another Send packet for that VI. By waiting for each acknowledgment, the end node does not have to worry about the network providing strict ordering and can choose an arbitrary source port, adaptive link set and destination port for each message.

b. The end node can restrict all the Send operations for a given VI to use the same source port, the same destination port, and a single adaptive path. By choosing only one path through the network, the end node is guaranteed that each Send packet it launches into the network will arrive at the destination in order.

The second approach requires the VIA end node to maintain state per VI that indicates which source port destination port and adaptive path is currently in use for that particular VI. Furthermore, the second approach allows the hardware to process descriptors in the higher performance NDAL mode.

With the second approach, Send packets from a single VI can stream onto the network without waiting for their accompanying acknowledgments. An incrementing sequence number is used so the destination node can detect missing, repeated, or unordered Send packets.

Ordering of RDMA Packets

RDMA operations have slightly different ordering requirements than Send operations. An RDMA packet contains the address to which the destination end node writes the packet contents. This allows multiple RDMA packets within an RDMA message to complete out of order. The contents of each packet are written to the correct place in the end node's memory, regardless of the order in which they complete.

RDMA request packets may be sent ordered or unordered. A bit in the packet header is set to 1 for ordered packets and is set to 0 for unordered packets. As will be explained later, this bit is used by the responder logic to determine if it should increment its copy of the expected sequence number. Sequence numbers do not increment for unordered packets. The end node is free to use different source ports, destination ports and adaptive paths for the packets. This freedom can be exploited for a performance gain through multipathing; simultaneously sending the RDMA packets of a single message across multiple paths.

When RDMA Read or Write packets are sent over a path that does not exhibit strict ordering with the Send packets from the same VI, care must be taken when launching packets for the following message. The next message cannot be started until the last acknowledgment of the RDMA Read or Write operation successfully completes.

In other words, when multipathing is used to generate RDMA Read or Write requests, the hardware must operate in the NDAA mode. This ensures the RDMA Read or Write is completed before moving on to subsequent descriptors.

An end node may choose to send RDMA packets strictly ordered. This can be advantageous for smaller RDMA transfers as the hardware can operate in NDAL mode. The VI can proceed to the next descriptor immediately after launching the last packet of a message that is sent strictly ordered (and hence used incrementing sequence numbers).

Ordering of Generated Response Packets at the Responder

The ServerNet II end node must respond to incoming Send requests and RDMA Write requests from a particular VI in strict order, and must write these packets to memory in strict order.

The ServerNet II end node must also respond to incoming RDMA Read requests from a particular VI in strict order.

Because response packets are transported by the network in strict order, the requestor will receive all incoming response packets for a particular VI in the same order as that in which the corresponding requests were generated.

VIA Message Sequence Numbers

The ServerNet SAN uses acknowledgment packets to inform the requestor that a packet completed successfully. Sequence numbers in the packets (and acknowledgments) are used to. allow the sender to support multiple outstanding requests to ensure adequate performance and to be able to recover from errors occurring in the network.

FIG. **6** is a graph depicting the generation, checking, and updating of VIA sequence numbers at requestor and responder nodes. In FIG. **6**, time increases in the downward direction. Requests are indicated by solid arrows directed to the right and responses by dotted arrows directed to the left.

Sequence Number Initialization

The requestor and responder logic each maintain an 8 bit sequence numbered for each VI in use. When the VI is created, the requester on one node and the respond on the remote node initialize their sequence numbers to a common value, zero i the preferred embodiment.

After this, the requester places its sequence number into each of the outgoing request packets. As depicted in FIG. **6**, the sequence number, SEQ, is included in each request packet. The responder compares the sequence number from the incoming request packet with the responder's local copy.

The responder uses this comparison to determine if the packet is valid, if it is a duplicate of a packet already received, or if it is an out-of-sequence packet. An out-of-sequence packet can only happen if the responder missed an incoming packet. The responder can choose to return a 'sequence error NACK packet' or it can simply ignore the out-of-sequence packet. In the latter case, the requester will have a time-out on the request (and presumably on the packet the responder missed) and initiate error recover. Generating a sequence error NACK packet is preferred as it forces the requester to start error recovery more quickly.

The following describes how the sequence numbers are generated and checked.

Generating Sequence Numbers for Request Packets

When transmitting ordered packets (i.e. transfers are on a specific source port to a specific destination port and the ACB specifies a specific lane) the request sequence number is incremented after each packet is sent. When transmitting unordered packets (i.e. multipathing is used and/or the ACB bits specify full link set adaptivity) the request sequence number is not incremented after such a packet is sent.

For example, in FIG. 6, during the first two Send transactions, the local copy of the request sequence number is incremented after the packet is sent (Rqst. SN=0 to 6). For the RDMA operation, which sends 2500 bytes unordered, the requester does not increment local copy of the request sequence number (Rqst. SN=6). The requester does not increment the local copy of the SN until after the first packet of the Send following the RDMA is transmitted.

Send packets are typically sent fully ordered lest the requestor have to wait for an acknowledgment for each packet before proceeding to the next. On the other hand, RDMA packets may be sent either ordered or unordered. To take advantage of multipathing, a requestor must use unordered RDMA packets.

The sender guarantees to never exceed the windowsize number of packets outstanding per VI. If S is the number of bits in the sequence number, then the windowsize is 2**(S-1).

A packet is outstanding until it and all its predecessors are acknowledged. The requestor does not mark a descriptor done until all packets requested by that descriptor are positively acknowledged.

Checking Sequence Numbers on Incoming Request Packets

The destination node responding to the incoming request packet checks each incoming request packet to verify its sequence number against the responder's local copy.

The responder logic compares its sequence number with the packet's sequence number to determine if the incoming packet is either:

- the expected packet it is looking for (i.e., the packet's sequence number is the same as the sequence number maintained by the responder logic), in which case the responder processes the packet and if all other checks are passed, the packet is Acknowledged and committed to memory. If the transaction is ordered, the responder increments its sequence number. If the transaction is unordered, the responder does not increment its sequence number;
- an out-of-sequence packet (which means an earlier incoming packet must have gotten lost), beyond the one it is looking for in which case the responder NACKs the packet and throws it away. The receive logic in the VI is not stopped and the responder does not increment its sequence number; or
- a duplicate packet (which is being resent because the requestor must not have received an earlier ACK) in which case the responder ACKs the packet and throws it away. If the request had been an RDMA Read, the responder completes the read operation and returns the data with a positive acknowledgment.

An example of the responder checking sequence numbers for ordered and unordered packets is given in FIG. 6. In FIG. 6, during the first two Send transactions, the responder checks that the SEQ in the packet matches the local copy of Rsp. SN. Since the Send packets include ACB indicating ordered packets, the Rsp. SN is incremented after each response packet is transmitted. At the end of the first two Send transactions, Rqst. SN and Rsp. SN both equal 6. The packets for the RDMA include an ACB indicating unordered receipt is allowed. Neither the requestor or responder increments its local copy of SN. Thus, at the end of the RDMA transaction both Rqst. SN and Rsp. SN=6. The first packet of the subsequent Send transaction has SEQ=6 and SEQ matches the local copy of Rsp. SN. Since Send packets are ordered, the responder increments its local copy of Rsp. SN.

Sequence Numbers on Response Packets

When generating either a positive or negative acknowledgment, the responder logic copies the incoming sequence number and uses it in the sequence number field of the acknowledgment.

The requestor logic matches incoming responses with the originating request by comparing the SourceID, VI number, Sequence number, transaction type, and transaction Serial Number (TSN) with that of the originating request.

Error Recovery and Path State

Error recovery is initiated by the requesting node whenever the requestor fails to get a positive acknowledgment for each of its request packets. A time-out or NACK indicating a sequence number error, can cause the requestor's Kernel Agent to start error recovery.

Error recovery involves three basic steps:

1) Completing a barrier operation(s) to flush out any errant request or response packets.

2) Disabling a bad path if the barrier operation failed.

3) Retransmitting from the earliest packet that had failed.

The first two steps will now be described with reference to FIG. 7, which is two interlocked state diagrams showing the state that software on the requestor and responder moves through for each path. In FIG. 7, dashed lines represent Kernel to Kernel Supervisory Protocol messages that modify the remote node's state.

The ServerNet architecture allows multiple paths between end nodes. The requestor repeats these two basic steps on each path until the packet is transmitted successfully.

The requestor and responder SW each maintain a view of the state of each path. The requestor uses its view of the path state to determine which path it uses for Send and RDMA operations. The responder uses its view of the path state to determine which input paths it allows incoming requests on. The responder logic maintains a four bit field (ReqInPath Vector) for each VI in use. Each of the four bits corresponds to one of the four possible paths between the requestor's two ports and the responder's two ports. The requestor only

accepts incoming requests from a particular source or destination port if the corresponding bit in the ReqInPathVector is set

The requestor and responder communicate using the kernel-to-kernel Supervisory protocol to communicate path state changes.

The requester's view of the path state transitions from good to bad whenever the requestor fails to get an acknowledgment (either positive or negative) to a request. The requestor detects the lack of an ACK or NACK by getting a time-out error. The requestor can attempt a barrier operation on the path to see if the failure is permanent or transient. If the barrier succeeds, the path is considered good and the original operation can be retried. If the barrier fails, the requestor must resort to a different good path.

Before the requestor can try a different good path, the requestor must inform the destination that the original path is bad. This is done by any path possible. For example, in VIA the Kernel Agent to Kernel Agent Supervisory Protocol is used. After the destination is informed the path is bad, the destination disables a bit in a four bit field (ReqInPath Vector), thereby ignoring incoming requests from that path. The requester then stops using the bad path until a subsequent barrier transaction determines that the path is good. After the destination acknowledges the supervisory protocol message, indicating that the destination has disabled requests from the offending path, the requestor is free to retry the message on a different path.

After a time-out error, the requester attempts to bring the path back to a useful state by completing a barrier operation. The barrier operation ensures there are no other packets in any buffer that might show up later and corrupt the data transfer.

Barrier operations are used in error recovery to flush any stale request or stale response packets from a particular path in the SAN. A path is the collection of ServerNet links between a specific port of two end nodes.

A VIA barrier operation is done with a RDMA Write followed by an RDMA Read. A number chosen from a large number space (either incrementing or pseudo random) is written to a fixed location (e.g. a page number agreed to, a priori, by the kernel agent-to-kernel agent Supervisory Protocol and either a fixed or random offset within the page). The number is then read back with an RDMA read. If the read value matches the write value, then the barrier succeeded and there are guaranteed to be no more Send or RDMA request or response packets on that path between the requester and responder.

If the RDMA operation fails because the number read back does not match the number written, then the barrier is tried again. This could have happened because a previous response in the network came back and fulfilled the barrier.

Note that the barrier needs to be done separately on all paths the RDMA operation could have taken. That is, if the RDMA operation was being generated from multiple source ports (multipathing) and was using full link adaptivity (the packets were allowed to take any one of four possible "lanes"), then separate barrier operations must be done from each source port to each destination port, over each of the possible link adaptive paths.

The barrier operation must be done for each of the possible "lanes" between a specific Source port and Destination port. A barrier done on one VI ensures that all other VIs using that source port and destination port have no remaining request or response packets lurking in the SAN.

If a path traverses a "fat-pipe" a separate barrier must be sent down each lane of the fat pipe. SW can either blindly send four barrier operations (one for each lane) or it can maintain state, telling how many lanes are in use on the fat pipes between any given source/destination pair. The barrier need only be sent along the same path as the original request that failed. There is no requirement for the barrier to be sent from or to the same VI.

Turning now to the third step, i.e., retransmitting from the earliest packet that failed to receive a positive acknowledgement.[, after] After notification of the error, requestor retransmits the packets starting at (or before) the packet that failed to receive a positive acknowledgement. The requestor can restart up to WindowSize number of packets. The responder logic acknowledges and then ignores any packets that are resent if they have already been stored in the receive queue. When the correct packet is reached, the responder logic can tell from the sequence number that it is now time to resume writing the data to the receive queue.

Examples of retransmission after failure to receive a response are depicted in FIGS. **8** and **9**. In FIG. **8**, the request packet with SEQ=2 is corrupted. The missing request is detected by the responder on the next packet and NACKed (Negative Acknowledged) and all subsequent packets are thrown away and NACKed. The requestor resets its send engine to start generating packets at the one that failed to receive an ACK (in this case Rqst. SN=2). The responder recognizes the SEQ=2 and accepts the packets.

In FIG. **9**, the response packet with SEQ=1 is corrupted. The missing response is detected when the requestor times out its transaction. The requestor resets its send engine to start generating packets at the one that failed to receive an ACK (in this case rqst.SN=1). The responder recognizes the resent packets as already having been received, acknowledges them and discards the data.

Note that the response packets for this particular RDMA transaction all have the same value of SEQ because the request SNs and response SNs are not incremented for RDMA transactions that are unordered. In this case, the TSNs are utilized by the requestor to match response packets to outstanding requests.

Error recovery places several requirements on the requestor's KA (Kernel Agent, the kernel mode driver code responsible for SAN error recovery):

1) The KA must determine the sequence number to restart with.

2) The KA must determine the proper data contents of the packet to be resent.

3) In order for the KA to determine the appropriate sequence number, it must be aware of how the hardware packetizes data under any given combination of descriptors, data segments, page crossings etc.

Note the responder side does not require KA involvement (unless a barrier operation fails).

The invention has now been described with reference to the preferred embodiments. Alternatives and substitutions will now be apparent to persons of skill in the art. For example, while the invention has been described in the context of the ServerNet II SAN, the principles of the invention are useful in any network that utilizes multiple paths between end nodes. Accordingly, it is not intended to limit the invention except as provided by the appended claims.

What is claimed is:

1. A method for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

creating a request packet for a request transaction,

routing the request packet from the source to the destination via a particular path;

maintaining at each of the source and destination a status for each of its respective paths;

detecting a time-out error for failure within a predetermined time limit to receive at the source an acknowledge (ACK) packet or a negative-acknowledge (NACK) packet in response to the request packet, the time-out error created from a failure of the particular path;

performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent;

periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent,

updating at the source the status for the particular path, and

routing to the destination information about the updated status via an alternate path to prompt updating at the destination of the status for the particular pat wherein a failed path is not used.

2. The method of claim **1**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

3. A method as in claim **1**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a link set.

4. A method as in claim **1**, wherein the request packet is discarded if it is invalid.

5. A method as in claim **1**, wherein the request packet contains a sequence number to indicate its place in a sequence of request packets and adaptive control bits to indicate the routing as ordered or not ordered.

6. A method as in claim **1**, further comprising:

maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value; and

including the sequence number in the request packet, wherein the sequence number is tracked by the source and destination, and wherein the NACK packet is created if a mismatch is found between the sequence number in the request packet and the sequence number at the destination.

7. A method for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value;

creating a request packet for a request transaction, the request packet containing the sequence number;

routing the request packet from the source to the destination via a particular path, wherein if the request transaction is ordered, the sequence number at the source is incremented;

maintaining at each of the source and destination a status for each of its respective paths;

checking the request packet for integrity and, if the request packet is valid, matching the sequence number in the packet with the sequence number at the destination;

creating an acknowledge (ACK) packet for a response transaction if the request packet is valid and the sequence number matching succeeds, and routing the ACK packet to the source, the ACK packet containing the sequence number from the request packet;

creating a negative acknowledge (NACK) packet for the response transaction if the sequence number matching fails, and routing the NACK packet to the source, the NACK packet containing the sequence number from the request packet;

incrementing the sequence number at the destination if the request transaction is ordered and the sequence number matching succeeds;

detecting a time-out error for failure within a predetermined time limit to receive at the source the ACK or the NACK packets in response to the request packet, the time-out error created from a failure of the particular path;

performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent;

periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent, updating a status at the source for the particular path and routing to the destination information about the updated status via an alternate path to prompt updating of the status for the particular path at the destination, wherein a failed path is not used.

8. A method as in claim **7**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of packets by selecting a single path as the particular path.

9. A method as in claim **7**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of packet routing by selection of paths from a path set.

10. A method as in claim **7**, wherein the request packet is discarded if it is invalid.

11. A method as in claim **7**, wherein the request packet contains adaptive control bits indicating whether the transaction is ordered or not ordered.

12. A system for error detection and recovery with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

means for creating a request packet for a request transaction; path means for routing the request packet from the source to the destination via a particular path;

means for maintaining at each of the source and destination a status for each of its respective paths;

means for detecting a time-out error for failure within a predetermined time limit to receive at the source an acknowledge (ACK) packet or a negative-acknowledge (NACK) packet in response to the request packet, the time-out error created from a failure of the particular path;

means for performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent, including means for periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

means for re-transmitting from the source the request packet via the particular path if the failure is transient; and

if the failure is permanent,

means for updating at the source the status for the particular path, and

means for routing to the destination information about the updated status via an alternate path to prompt updating at the destination of the status for the particular path, wherein a failed path is not used.

13. The system of claim **12**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

14. A system as in claim **12**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a link set.

15. A system as in claim **12**, further comprising:

means for discarding the request packet if it is invalid.

16. A system as in claim **12**, wherein the request packet contains a sequence number to indicate its place in a sequence of request packets and adaptive control bits to indicate the routing as ordered or not ordered.

17. A system as in claim **12**, further comprising:

means for maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value; and

means for including the sequence number in the request packet, wherein the sequence number is tracked by the source and destination, and wherein the NACK packet is created if a mismatch is found between the sequence number in the request packet and the sequence number at the destination.

18. A system for error detection and recovery in a system with a plurality of networked nodes, including source and destination, communicating with each other via paths, comprising:

means for maintaining a sequence number in each of the source and destination, the sequence numbers being initialized to a common value;

means for creating a request packet for a request transaction, the request packet containing the sequence number;

means for routing the request packet from the source to the destination via a particular path, including means for incrementing the sequence number at the source if the request transaction is ordered;

means for maintaining at each of the source and destination a status for each of its respective paths;

means for checking the request packet for integrity, including means for matching the sequence number in the packet against the sequence number at the destination if the request packet is valid;

means for creating an acknowledge (ACK) packet for a response transaction if the request packet is valid and a sequence number matching succeeds, including means for routing the ACK packet to the source, the ACK packet containing the sequence number from the request packet;

means for creating a negative acknowledge (NACK) packet for the response transaction if the sequence number matching fails, including means for routing the NACK packet to the source, the NACK packet containing the sequence number from the request packet;

means for incrementing the sequence number if the request transaction is ordered and the sequence number matching succeeds;

means for detecting a time-out error for failure within a predetermined time limit to receive at the source the ACK or the NACK packets in response to the request packet, the time-out error created from a failure of the particular path;

means for performing a barrier transaction via the particular path to determine if the failure of the particular path is transient or permanent, including by periodically repeating the barrier transaction via the particular path in order to determine if its failure is cured;

means for re-transmitting the request packet via the particular path if the failure is transient; and

if the failure is permanent,

means for updating a status at the source for the particular path, and

means for routing to the destination information about the updated status via an alternate path to prompt updating of the status for the particular path at the destination, wherein a failed path is not used.

19. A system as in claim **18**, wherein when the transaction is ordered the routing is deterministic preserving strict ordering of request, ACK and NACK packets by selecting a single path as the particular path.

20. A system as in claim **18**, wherein when the transaction is not ordered the routing is link set adaptive enabling dynamic change of request, ACK and NACK packet routing by selection of paths from a path set.

21. A system as in claim **18**, further comprising:

means for discarding the request packet if it is invalid.

22. A system as in claim **18**, wherein the request packet contains adaptive control bits indicating whether the transaction is ordered or not ordered.

* * * * *